OPTIMISTIC EXECUTION AND CHECKPOINT COMPARISON FOR ERROR RECOVERY IN PARALLEL AND DISTRIBUTED SYSTEMS [1]

Junsheng Long and W. Kent Fuchs

Coordinated Science Laboratory
1101 W. Springfield Ave.
University of Illinois
Urbana, IL 61801

Jacob A. Abraham

Computer Engineering Research Center
2201 Donley Dr., Suite 395
University of Texas at Austin
Austin, TX 78758

Principal contact: Junsheng Long

long@crhc.uiuc.edu
(217) 333-8294
FAX: (217) 244-5686

May 8, 1992

Abstract

This paper describes a checkpoint comparison and optimistic execution technique for error detection and recovery in distributed and parallel systems. The approach is based on lookahead execution and rollback validation. It uses replicated tasks executing on different processors for forward recovery and checkpoint comparison for error detection. Two schemes derived from this strategy are analyzed and compared with triplication and voting, and with two common backward recovery methods. The impact of checkpoint time, checkpoint validation time, and process restart time is also examined. An implementation on a Sun NFS network with six benchmark programs is presented. Compared with classic checkpointing and rollback techniques, our strategy provides rapid recovery and requires, on average, fewer processors than standard replication and voting methods. This strategy is useful in systems where spare processors are available at the time of recovery.

Key Words: *fault tolerant computing, checkpointing, error detection, and error recovery.*

[1] This research is supported in part by the National Aeronautics and Space Administration (NASA) under Contract NAG 1-613 and in part by the Department of the Navy and managed by the Office of the Chief of Naval Research under Contract N00014-91-J-1283.

Earlier versions of portions of this work were presented at ICPP-90 and the IFIP Working Conference on Dependable Computing for Critical Applications, 1991.

I. INTRODUCTION

Considerable research has been devoted to checkpoint-based backward recovery schemes [1–5]. There have also been techniques proposed which combine replication with voting and checkpoint rollback recovery [6–8]. The RAFT algorithms replicate the computation on two processors to achieve error detection and rollback recovery [6,7]. If the results produced by the replicated tasks do not match, the task is executed on other processors until a pair of matched results is found. Checkpoint-based backward recovery has two drawbacks: an execution time penalty due to checkpointing and rollback, and the problem of determining if a checkpoint is error free. Although placing checkpoints optimally can reduce the execution time penalty to some extent, the computation lost by rollback is inherent [2–5]. One approach to validating a checkpoint is to validate the system state via concurrent error detection or system diagnosis, before a checkpoint is taken [9,10]. Another is to simply keep a series of consecutive checkpoints and perform multiple rollbacks when necessary [11]. In contrast to backward recovery, forward recovery attempts to reduce the lost computation by manipulating some portion of the current state to produce an error-free new state. However, forward recovery generally depends on accurate damage assessment, a correction mechanism, and sometimes massive redundancy (e.g., NMR) [1,12].

In this paper, we present a checkpoint-based error detection and optimistic recovery strategy for parallel and distributed systems. This strategy requires neither application-specific error correction nor massive static NMR for error masking to achieve forward recovery. It uses checkpoint comparison for checkpoint validation and optimistic execution for forward recovery.

The following section describes our recovery strategy; the subsequent section discusses the performance evaluation of the recovery schemes derived from our strategy. Section IV presents an experimental evaluation with a distributed implementation on a Sun NFS-based network.

II. Recovery with Optimistic Execution Using Checkpoints

A. Computation and System Model

The system considered consists of homogeneous processing elements connected to each other and to secondary storage by a network. The processing element can be either a computer node in a distributed system or a CPU node in a multiprocessor system. The network can be an LAN for a distributed system or a general connection network for a parallel system. We assume that the necessary checkpoints are retained on a reliable second y storage and are accessible through the interconnection network.

A task is an independent computation and it can be a group of related subtasks. A task is divided into a series of sequential subcomputation sessions by checkpoints. A process is the task running on a processing element. A process can be replicated on different processors.

A checkpoint consists of two types of information: the current process state for process restart and the test information for process state validation. The state and test information may or may not be separate entities within the checkpoint. If the checkpoint is the complete run-time process image, the test information can be the image itself or the signature of the image. Checkpoint comparison is used to detect erroneous system state or validate the checkpoint. This implies that the probabilities of the two checkpoints being identical as a result of one or two erroneous processes are negligible.

This paper only deals with the faults that cause an error in a process and result in an erroneous checkpoint. Faults in the processor interconnection network or the secondary storage may not be detectable nor recoverable in our approach. We also assume a reliable (voter) process that performs checkpoint comparisons.

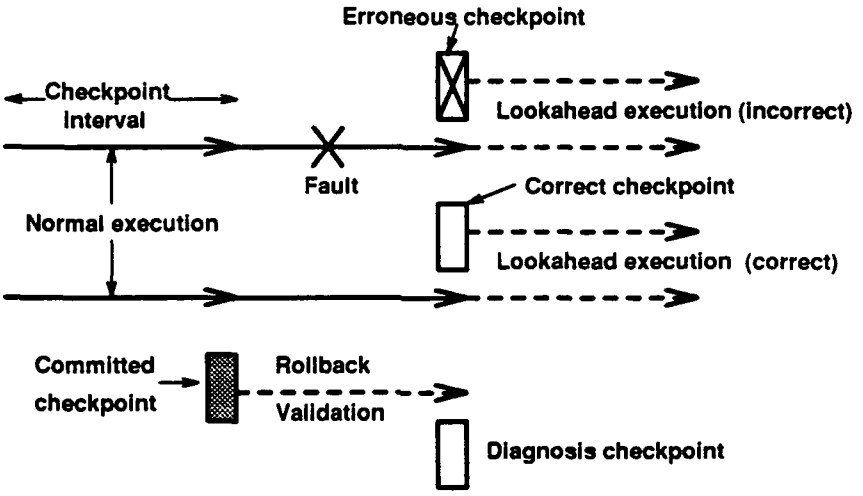

Figure 1. Lookahead Execution and Rollback Validation.

B. Lookahead Execution and Rollback Validation

Two essential features of our recovery strategy are lookahead execution to reduce the computation loss due to recovery and rollback validation to diagnose the correctly scheduled lookahead. These concepts are illustrated in Figure 1. As in the RAFT scheme, a task is replicated and executed concurrently on two different processors [6, 7]. At the end of one computation session, two checkpoints are produced by the replicated process pair. A voter process compares the newly generated but uncommitted checkpoints to determine if the process state is error free. If the two checkpoints are identical, the system state is valid. Either of the checkpoints can be committed for the past computation session and the process pair advances to the next session.

If the uncommitted checkpoints disagree, then the checkpoints contain an erroneous state. Instead of rolling back, two identical task processes are started from the uncommitted checkpoints on two additional processors. This optimistic scheduling is called lookahead (optimistic) execution. Meanwhile, another process rolls back to the last committed checkpoint on a fifth processor. After

a checkpoint interval, Δ, a diagnosis checkpoint is produced by the rollback (validation) process. This checkpoint is compared to the two disagreeing (uncommitted) checkpoints. If there is a match, the error-free checkpoint is identified and committed. The process pair that was executed ahead from the disagreeing erroneous checkpoint and the rollback validation process are terminated. In this strategy, the two additionally scheduled lookaheads make it possible not to roll back the whole system when there is an error during lookahead executions. In this case, the lookahead pair from the new verified checkpoint is treated as the normal pair. This pair can start a new round of lookahead and rollback validation without rolling back the whole system.

Compared to the static redundancy of three processors for TMR, this strategy uses two processors for the common error-free situation and a dynamic redundancy of five for the rare occurrence of an error. The potential for forward recovery lies in the fact that there should be at least one correct process (thus, one valid checkpoint) during the normal run, since the lookahead execution from this valid checkpoint advances the computation without rollback. However, rollback may not be avoidable when the diagnosis checkpoint does not agree with either of the two uncommitted disagreeing checkpoints, since all optimistic executions may be incorrect.

C. Recovery Design Considerations

With respect to lookahead and rollback scheduling, there are four critical parameters in designing a specific recovery scheme based on the approach described. The first is the number of replicated processes in the normal run, which we call *base (redundancy) size*. The larger the base size, the more potential there is for forward recovery, since it is likely to have an error-free checkpoint for successful lookaheads. The second is the *validation size* or the number of processes used for rollback validation; the third is the *validation depth* or the number of retries of the rollback

validation process if a rollback validation fails to diagnose the disagreeing checkpoints. We can use either a larger validation size or a larger validation depth to increase the diagnosis success rate. In the case of a larger validation depth, the rollback validation success rate is increased by using time redundancy. The fourth is the *lookahead size* or the number of lookahead processes scheduled.

A forward recovery scheme is recursive if its validation depth is unlimited. In this case, the processes executed ahead can spawn their children of lookahead and validation tasks unboundedly, as the validation retries increase. If multiple failures during a checkpoint period are rare, then it is unlikely that recursive validation and lookahead process spawning will be required. A nonrecursive scheme is an approximation of its recursive counterpart. In fact, it is the corresponding recursive scheme with all validation re-tries greater than the validation depth truncated. If the processor resource is limited, limiting the lookaheads scheduled (lookahead size $<$ base size) leads to graceful performance degradation [13]. At one extreme, the recovery scheme degenerates into a normal rollback scheme such as RAFT if the lookahead size is zero [6, 7].

D. Limitations

Our approach is useful in systems where extra spare processors are available at the time of recovery. Other limitations to our approach are the requirement that an error results in an erroneous checkpoint, and the implementation requirement that the checkpoints generated from different processors for the same computation should be identical if there is no error.

III. Performance Evaluation

In our analytical and experimental evaluation, three types of overhead are considered: checkpoint time (t_k), process restart time (t_r) and checkpoint testing time (t_t). For purposes of analysis,

constant checkpoint intervals and overheads are used. Each processor has a constant probability of failure, p_f, during one computation session $(\Delta + t_k)$ with or without restart and checkpoint test. This assumption implies two requirements. The first is a Poisson distribution for the failure distribution, while the second is $t_t \ll \Delta + t_k$ and $t_r \ll \Delta + t_k$ since the probability of failure over $[0,\Delta + t_k]$ is required to be equal to that over $[0,\Delta + t_k + t_t + t_r]$. The typical test time t_t and restart time t_r are in the order of a fraction of a second and the checkpoint interval Δ on the order of minutes or hours.

In order to consider the impact of the centralized file server that handles checkpoint files, we assume that t_k and t_r are approximately n-fold, when the n processes access their checkpoint files at the same time. This assumption enables us to study the impact of a file server by adjusting t_k and t_r, since both restart time t_r and checkpoint time t_k will be increased due to the file accesses to a single server. The increase in t_r and t_k may not be proportional to the number of processes that access the same file. However, a checkpoint file usually contains many blocks. A fair server policy guarantees that the n processes finish their access to the checkpoint file at approximately the same time. We also assume that checkpoint comparison is performed by a voter process on a host that can access the file system locally, and thus t_t is not changed.

A. Performance Metrics

The performance measures we examine in this paper are:

- *Relative Execution Time*, R_e: the ratio of the expected execution time (T_e) over the error-free execution time (T_0). This measure normalizes the effect of the execution time for different computations. If R_e is close to one, the execution time will be close to the error-free execution time.

- *Number of Processors*, N_p: the average number of processors used by the system or $\frac{1}{T_e}\int_0^{T_e} N_p(t)dt$. It describes the processor redundancy required by a recovery scheme. The maximal instantaneous $N_p(t)$ reflects the maximal processor requirement.

- *Number of Checkpoints*, N_c: the average number of checkpoints stored in the system or $\frac{1}{T_e}\int_0^{T_e} N_c(t)dt$. The maximal instantaneous $N_c(t)$ reflects the maximal storage requirement.

B. Alternative Recovery Schemes

We examine two alternative schemes derived from our proposed recovery strategy and three other common schemes. These five recovery schemes are characterized in Table 1. Both DMR-F-1 and DMR-F-2 are nonrecursive schemes derived from our recovery strategy. Their rollback validation is limited to one try with one or two rollback validation processes. The TMR-F scheme is the common TMR forward recovery scheme using error masking and majority voting. The DMR-B-1 and DMR-B-2 schemes are recursive rollback schemes modified from the RAFT algorithms [6, 7]. They use two processes for the normal execution. If the checkpoints match after checkpointing, the execution advances to the next session. If there is no match, one or two validation processes roll back repeatedly until a matched checkpoint pair is obtained. These two recursive rollback schemes have the best performance among all their nonrecursive approximations.

C. Discussions

The derived analytical model for DMR-F-1 and DMR-F-2 is presented in Appendix A. The results are summarized in Tables 2 and 3 (the symbol, *fs*, denoted for the corresponding cases with a centralized file server). The analysis for TMR-F, DMR-B-1 and DMR-B-2 is very similar [13]. Generally, R_e, N_p or N_c can be expressed in terms of the relative overhead factors as

Table 1. Five Schemes Compared.

DMR-F-1:	a nonrecursive forward recovery scheme with base size = 2, validation size = 1, validation depth = 1 and lookahead size = 2.
DMR-F-2:	a nonrecursive forward recovery scheme with a base size = 2 validation size = 2, validation depth = 1 and lookahead size = 2.
TMR-F:	a triple module redundancy forward recovery scheme (base size = 3) with validation size = 0, validation depth = 0 and lookahead size = 0.
DMR-B-1:	a recursive backward recovery scheme with base size = 2, validation size = 1, validation depth = ∞ and lookahead size = 0.
DMR-B-2:	a recursive backward recovery scheme with base size = 2, validation size = 2, validation depth = ∞ and lookahead size = 0.

$$m = c + \alpha + \beta\frac{t_r}{\Delta + t'_k} + \gamma\frac{t_t}{\Delta + t'_k} + \delta\frac{t'_k}{\Delta + t'_k},$$

where c is a constant, $m \in \{R_e, N_c, N_p\}$ and t'_k is either $2t_k$ or $3t_k$. The constant c is the error-free part of the performance measure, while α is the performance degradation due to rollbacks in the schemes we considered. The smaller α is for a recovery scheme, the more effective it is in terms of reducing the execution time degradation. In this paper, α is called the coefficient of overhead due to rollback. This rollback overhead can not be eliminated and depends only on the failure probability, p_f. The expression $c + \alpha$ represents the inherent performance of a particular scheme. The factors of β, γ, and δ are the overhead coefficients for process restart, checkpoint comparison and checkpointing.

For R_e, the overhead coefficients, α, β, γ and δ are related only to p_f. For N_p and N_c, the coefficients also include a factor of $\frac{1}{R_e}$. Normally, the relative overheads are very small, and we can approximate R_e with the zero-overhead R_e. This approximation, in fact, gives the upper bound for α, β, γ and δ in N_p and N_c, since the presence of t_r, t_t and t_k increases R_e. Therefore, N_p and N_c are approximately a linear function of overhead factors. The overhead coefficients represent the

Table 2. Analytical Evaluation Summary: DMR-F-1.

p_l	$2p_f(1-p_f)^2$
p_r	$2p_f^2(1-p_f)+p_f^2$
T_e	$n(\Delta+t_k)\left(1+\frac{2p_r}{1-p_r}\right)+nt_r\frac{p_l+2p_r}{1-p_r}+nt_t\frac{2.5p_l+3p_r}{1-p_r}$
R_e	$1+\frac{2p_r}{1-p_r}+\frac{p_l+2p_r}{1-p_r}\frac{t_r}{\Delta+t_k}+\frac{2.5p_l+3p_r}{1-p_r}\frac{t_t}{\Delta+t_k}$
N_c	$1+2\frac{p_l+p_r}{(1-p_r)R_e}+2\frac{p_l+p_r}{(1-p_r)R_e}\frac{t_r}{\Delta+t_k}+2\frac{6.25p_l+8p_r}{(1-p_r)R_e}\frac{t_t}{\Delta+t_k}$
$max(N_c)$	8
N_p	$2+3\frac{p_l+p_r}{(1-p_r)R_e}+3\frac{p_l+p_r}{(1-p_r)R_e}\frac{t_r}{\Delta+t_k}+3\frac{1.5p_l+2p_r}{(1-p_r)R_e}\frac{t_t}{\Delta+t_k}$
$max(N_p)$	5
$R_e(fs)$	$1+\frac{2p_r}{1-p_r}+\frac{p_l+5/3p_r}{1-p_r}\frac{3t_r}{\Delta+2t_k}+\frac{2.5p_l+3p_r}{1-p_r}\frac{t_t}{\Delta+2t_k}+\frac{p_l+p_r}{1-p_r}\frac{3t_k}{\Delta+2t_k}$
$N_c(fs)$	$1+2\frac{p_l+p_r}{(1-p_r)R_e(fs)}+2\frac{p_l+p_r}{(1-p_r)R_e(fs)}\frac{3t_r}{\Delta+2t_k}+$ $+2\frac{6.25p_l+8p_r}{(1-p_r)R_e(fs)}\frac{t_t}{\Delta+2t_k}+2\frac{p_l+p_r}{(1-p_r)R_e(fs)}\frac{3t_k}{\Delta+2t_k}$
$N_p(fs)$	$2+3\frac{p_l+p_r}{(1-p_r)R_e}+3\frac{p_l+p_r}{(1-p_r)R_e}\frac{3t_r}{\Delta+2t_k}+$ $+3\frac{1.5p_l+2p_r}{(1-p_r)R_e}\frac{t_t}{\Delta+2t_k}+3\frac{p_l+p_r}{(1-p_r)R_e}\frac{3t_k}{\Delta+2t_k}$

contribution of their corresponding overhead factors to the performance degradation. The larger the coefficient for an overhead factor, the more important this overhead factor is with respect to performance degradation.

For the noncentralized file server situation, δ is zero. The checkpoint time, t_k, does not appear as an overhead factor because an error-free execution time that includes checkpoint time, $n(\Delta+t_k)$, is used as the base for our performance measures. The overhead coefficient for the checkpoint time is $c+\alpha$ if the checkpoint-less error-free execution time is used as the base for the performance measures. For example, R_e can be redefined as the ratio of the expected execution time over the checkpointless error-free execution time, $\frac{T_e}{n\Delta}$, instead of $\frac{T_e}{\Delta+t_k}$. That is,

Table 3. Analytical Evaluation Summary: DMR-F-2.

p_l	$2(1-p_f)p_f(1-p_f^2)$
p_s	$p_f^2(1-p_f)^2$
p_r	$2p_f^3(1-p_f)+p_f^2(2(1-p_f)p_f+p_f^2)$
T_e	$n(\Delta+t_k)\left(1+\frac{p_s+2p_r}{1-p_r}\right)+nt_r\frac{p_l+2p_s+2p_r}{1-p_r}+nt_t\frac{3.5p_l+5p_s+5p_r}{1-p_r}$
R_e	$1+\frac{p_s+2p_r}{1-p_r}+\frac{p_l+2p_s+2p_r}{1-p_r}\frac{t_r}{\Delta+t_k}+\frac{3.5p_l+5p_s+5p_r}{1-p_r}\frac{t_t}{\Delta+t_k}$
N_c	$1+2\frac{p_l+p_s+p_r}{(1-p_r)R_e}+2\frac{p_l+p_s+p_r}{(1-p_r)R_e}\frac{t_r}{\Delta+t_k}+2\frac{11p_l+21.5p_s+21.5p_r}{(1-p_r)R_e}\frac{t_t}{\Delta+t_k}$
$max(N_c)$	9
N_p	$2+4\frac{p_l+p_s+p_r}{(1-p_r)R_e}+4\frac{p_l+p_s+p_r}{(1-p_r)R_e}\frac{t_r}{\Delta+t_k}+4\frac{2.5p_l+5.5p_s+5.5p_r}{(1-p_r)R_e}\frac{t_t}{\Delta+t_k}$
$max(N_p)$	6

$R_e(fs)$	$1+\frac{p_s+2p_r}{1-p_r}+\frac{p_l+5/3p_s+5/3p_r}{1-p_r}\frac{3t_r}{\Delta+2t_k}+$ $+\frac{3.5p_l+5p_s+5p_r}{1-p_r}\frac{t_t}{\Delta+2t_k}+\frac{p_l+p_s+p_r}{1-p_r}\frac{3t_k}{\Delta+2t_k}$
$N_c(fs)$	$1+2\frac{p_l+p_s+p_r}{(1-p_r)R_e(fs)}+2\frac{p_l+p_s+p_r}{(1-p_r)R_e(fs)}\frac{3t_r}{\Delta+2t_k}+$ $+2\frac{11p_l+21.5p_s+21.5p_r}{(1-p_r)R_e(fs)}\frac{t_t}{\Delta+2t_k}+2\frac{p_l+p_s+p_r}{(1-p_r)R_e(fs)}\frac{3t_k}{\Delta+2t_k}$
$N_p(fs)$	$2+4\frac{p_l+p_s+p_r}{(1-p_r)R_e}+4\frac{p_l+p_s+p_r}{(1-p_r)R_e}\frac{3t_r}{\Delta+2t_k}+$ $+4\frac{2.5p_l+5.5p_s+5.5p_r}{(1-p_r)R_e}\frac{t_t}{\Delta+2t_k}+4\frac{p_l+p_s+p_r}{(1-p_r)R_e}\frac{3t_k}{\Delta+2t_k}$

$$R'_e = \frac{T_e}{n\Delta} = \frac{T_e}{\Delta + t_k}\frac{\Delta + t_k}{\Delta} = R_e\frac{\Delta + t_k}{\Delta} = c + \alpha + (c+\alpha)\frac{t_k}{\Delta} + \beta\frac{t_r}{\Delta} + \gamma\frac{t_t}{\Delta}.$$

Checkpointing overhead is inherent in any checkpoint-based scheme since checkpoint time is always included in the execution whether a fault occurs or not. To minimize the impact of this overhead, the optimal checkpoint interval or frequency should be utilized.

The performance degradation due to a centralized file server is reflected in two ways. The first is the increased overhead caused by the file access serialization. For example, an approximate factor of 3 appears for the restart overhead term, $\frac{t_r}{\Delta+t_k}$ in $R_e(fs)$. The second is the nonzero overhead coefficient for checkpoint time (δ) because of the extra checkpointing activities by the lookahead and rollback validation processes during recovery.

D. Comparison

In order to compare the five schemes we described above, R_e, N_p, and N_c are plotted in Figures 2, 3 and 4. The solid curves depict the zero-overhead case (i.e., the inherent performance, $c + \alpha$), whereas the dotted curves depict the case with 5% overheads (e.g., t_k, t_r and t_t are 5% of $\Delta + t_k$, respectively).

In Figure 2, the expected execution times for DMR-F-1 and DMR-F-2 are comparable to that for TMR-F. In fact, their execution time is nearly the same as the error-free execution time. The execution times for the rollback schemes (DMR-B-1 and DMR-B-2) can be as high as 20 percent more than the error-free execution time. The increase in R_e with p_f shows that rollback is still possible in TMR-F, DMR-F-1 and DMR-F-2, even though these schemes can perform forward recovery. For DMR-F-1, R_e is larger than that for DMR-F-2 because there are more rollback validation failures in DMR-F-1.

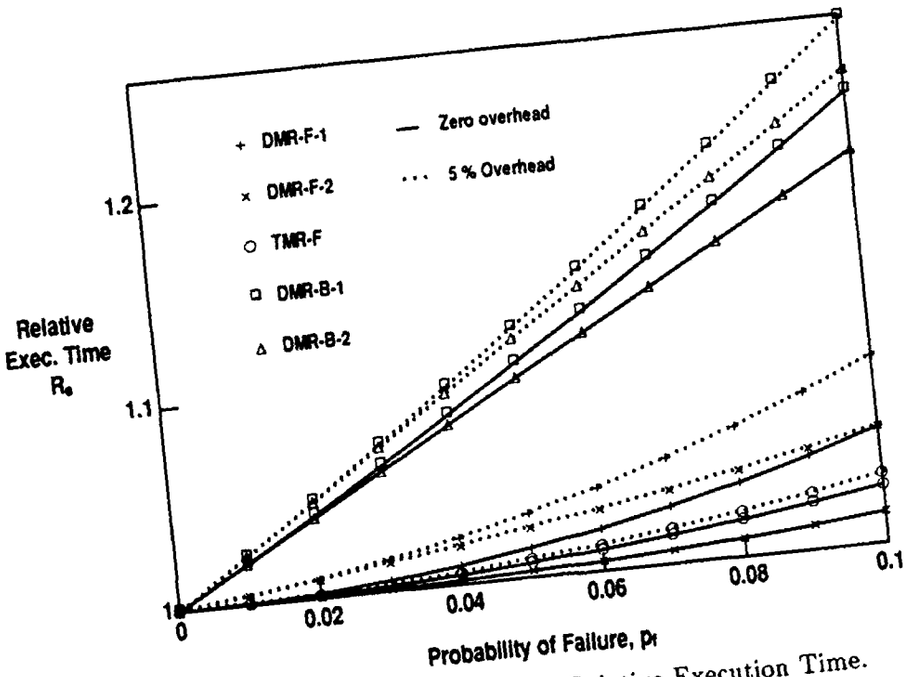

Figure 2. Comparison: Relative Execution Time.

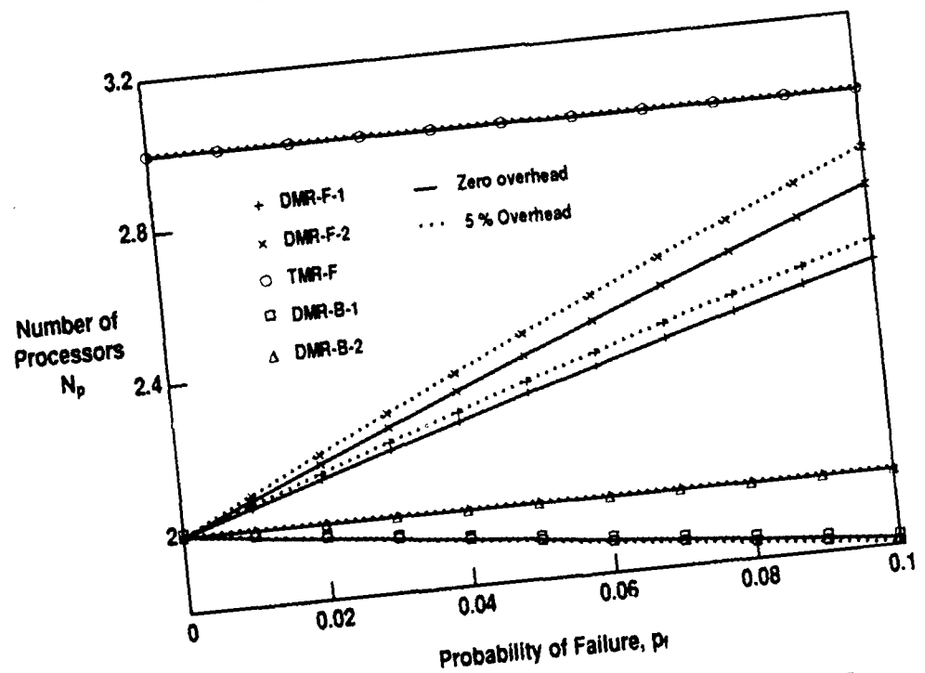

Figure 3. Comparison: Number of Processors.

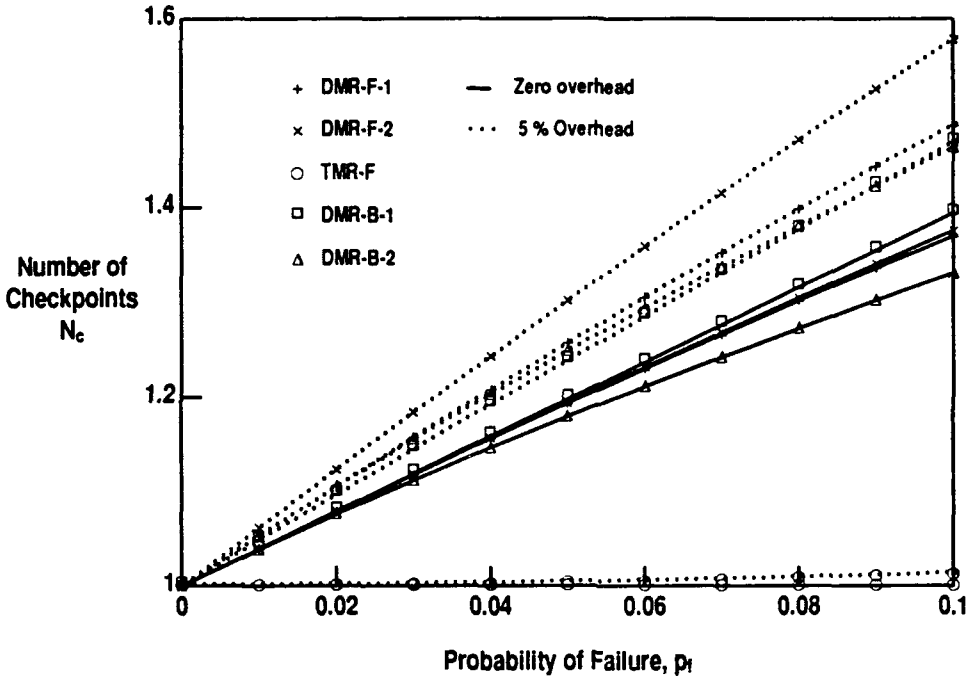

Figure 4. Comparison: Number of Checkpoints.

The average number of processors used for DMR-F-1 and DMR-F-2 is less than that of TMR (Figure 3). Using more than three processors dynamically for the infrequent error situation enables DMR-F-1 and DMR-F-2 to reduce the overall processor redundancy. As expected, the rollback schemes, DMR-B-1 and DMR-B-2, use on average fewer processors than the others. For DMR-B-1, N_p decreases with p_f because only one processor is used during recovery.

The number of checkpoints increases with p_f for all schemes except TMR-F. For TMR-F, N_c is close to one. For DMR-F-1 and DMR-F-2, N_c is slightly higher than that for DMR-B-1 and DMR-B-2. It seems contradictory to the fact that more checkpoints would be accumulated during recovery for DMR-B-1 and DMR-B-2. However, DMR-B-1 and DMR-B-2 do have a smaller N_c than DMR-F-1 and DMR-F-2 because they have a longer execution time than DMR-F-1 and DMR-F-2 due to rollbacks. The difference in N_c may be insignificant, since most modern systems usually have a large secondary storage for the checkpoint files.

As expected, the presence of overhead increases R_e. Both DMR-F-1 and DMR-F-2 still have

an execution time close to the error-free execution time (within 5% for DMR-F-2 and 10% for DMR-F-1). For DMR-F-1 and DMR-F-2, N_p is increased less than 1% because the extra processors are used only during recovery. N_p for TMR-F and DMR-B-2 are constant, since they always use three and two processors, respectively, during both normal execution and recovery.

E. Overhead Impact on Performance

The impact of the checkpoint overhead is determined by $c + \alpha$ and depicted in Figures 2, 3 and 4 as the zero-overhead curves. The impact of checkpoint overhead on R_e for DMR-F-1, DMR-F-2 and TMR-F is smaller than that for DMR-B-1 and DMR-B-2. This is because the rollback reduction in DMR-F-1, DMR-F-2 and TMR-F leads to fewer checkpointing sessions in computation (Figure 2). For DMR-F-1 and DMR-F-2, N_p is more sensitive to the checkpoint overhead than that for TMR-F, DMR-B-1 and DMR-B-2 as indicated by a positive slope in Figure 3. The static redundancy employed in TMR-F and DMR-B-2 is reflected by the flat slopes in Figure 3. Except for TMR-F, the sensitivity of N_c to the checkpoint overhead is reflected by the relatively steep slopes in Figure 4.

Figures 5, 6 and 7 compare the overhead coefficients for restart time and checkpoint comparison time. The solid curves represent the impact of t_r; the dotted ones depict the impact of t_t. The impact of the comparison time t_t is more than twice that of the restart time t_r. This suggests that any decrease in comparison time will result in a bigger gain in performance improvement than will an equal decrease in restart time. In Figure 5, t_r and t_t affect R_e for DMR-F-2 and DMR-B-2 more than R_e for other schemes; TMR-F is insensitive to both t_r and t_t. For DMR-F-1 and DMR-F-2, t_r and t_t affect N_p more than for other schemes because both schemes employ extra processors during recovery (Figure 6). As indicated by the large slopes of the dotted curves in Figure 7, the number

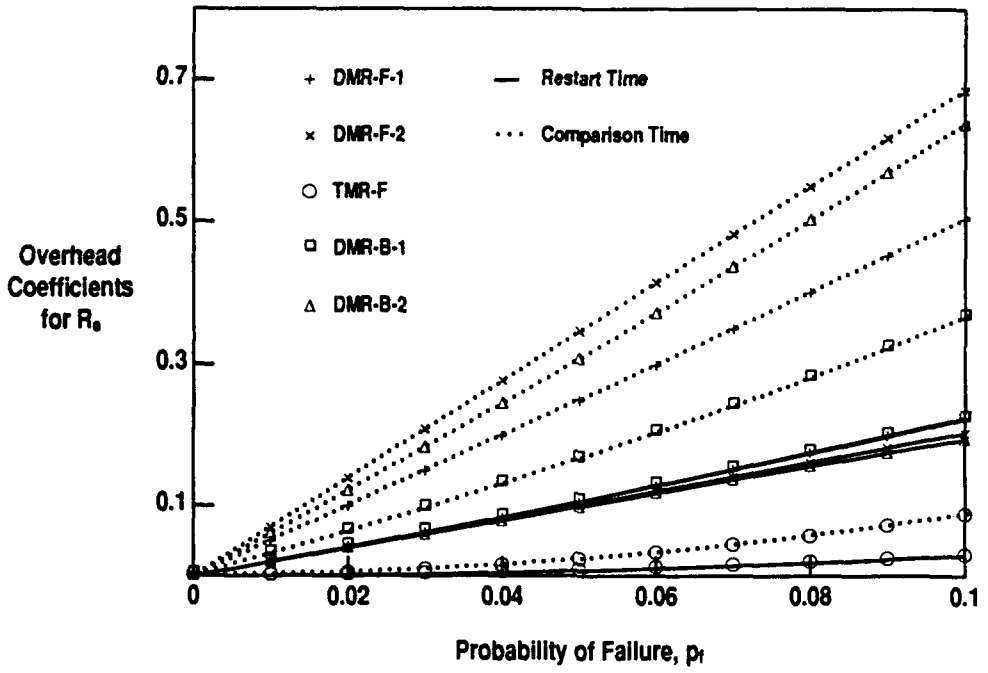

Figure 5. Overhead Impact on Execution Time.

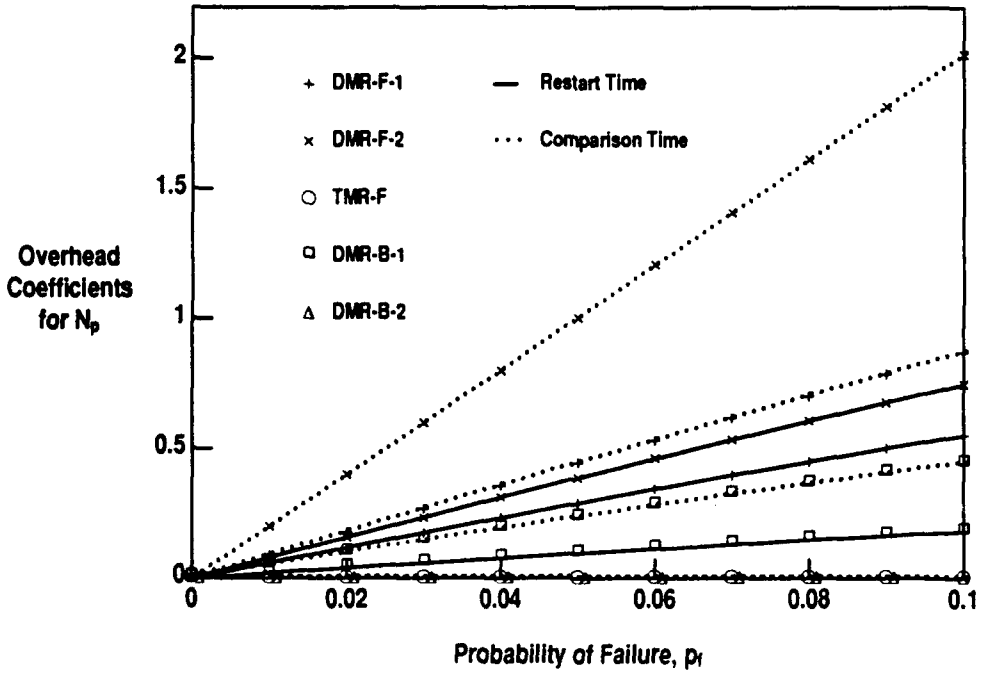

Figure 6. Overhead Impact on Number of Processors.

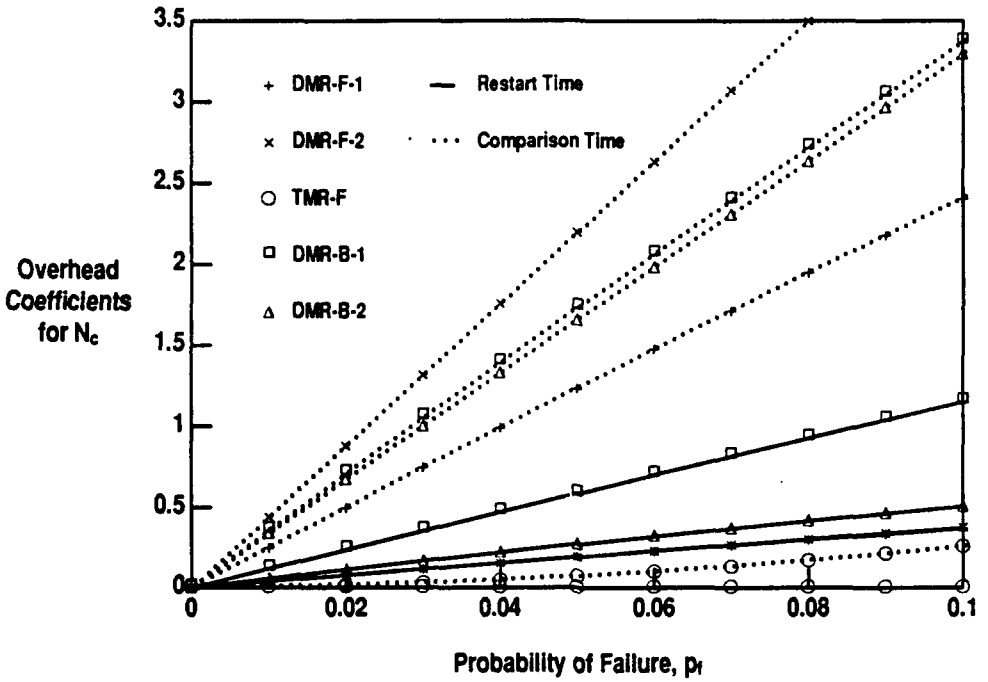

Figure 7. Overhead Impact on Number of Checkpoints.

of checkpoints for all schemes except TMR-F is sensitive to the checkpoint comparison overhead.

The impact of a centralized file server is depicted in Figure 8 for a case with 5 % overheads. The solid curves are for the centralized server case, while the dotted ones for the noncentral server case. The impact of a single file server for TMR-F, DMR-B-1, and DMR-B-2 is not as significant as that for DMR-F-1 and DMR-F-2, since there are additional checkpoint operations and restarts by the lookahead and rollback validation processes during recovery for DMR-F-1 and DMR-F-2.

F. Checkpoint Placement

The formulas for T_e in Tables 2 and 3 can be used to minimize the impact of checkpoint time on execution time by selecting the proper checkpoint interval or frequency. Figure 9 shows the expected execution time under different failure rates and overhead costs for DMR-F-1. The optimal checkpoint frequency can be obtained by either numerical or graphical means, given a failure rate, task computation time, and overhead costs such as checkpoint time, restart time, and comparison

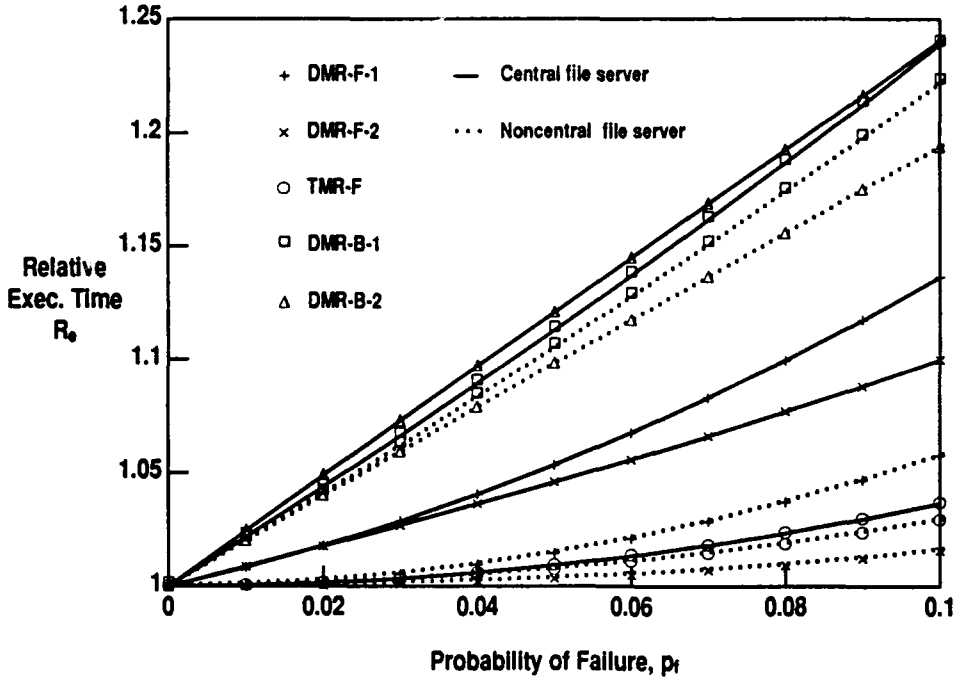

Figure 8. Impact of a Central File Server on Execution Time.

time.

Note in Figure 9 that for a low checkpointing overhead, the execution time curve near the bottom is rather flat. This suggests that an exact checkpoint interval is not necessary since a few additional checkpoints still give a near optimal solution. For small failure rates, the checkpoint interval is usually large or checkpoint frequency is small. This observation agrees with the previous studies on optimal checkpoint placement for other recovery schemes [2-5].

IV. Experimental Implementation Evaluation

A. Distributed Implementation

In this section, we discuss our DMR-F-1 implementation for a distributed system consisting of a Sun 3/280 server and a pool of 12 Sun 3/50 diskless workstations. The server provides a Sun NFS transparent access to remote file systems under SunOS 4.0. A voter task for the checkpoint comparison and recovery initiation runs on this server. All checkpoints are kept by the server. The

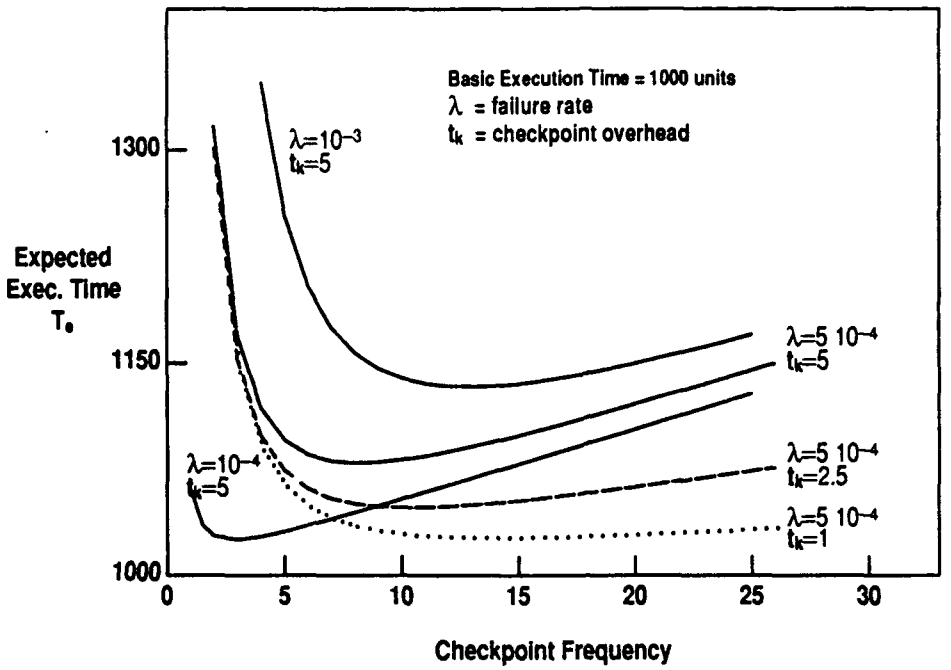

Figure 9. Optimal Checkpoint Placement.

Sun 3/50 workstations are used as the processing units. This setting makes it possible to evaluate the impact of the centralized file server. Our implementations are entirely user level with no kernel modifications required.

A checkpoint used in our implementation is a snapshot of a process run-time image at the time of checkpointing. There has been considerable research concerning checkpoint construction in UNIX [14–17]. Smith implemented a mechanism for checkpoint construction in UNIX for the purpose of process migration [14]. His checkpoint is an executable file generated by a checkpoint operation. It contains the text segment, the data segment, as well as the stack segment of the process state. The stack segment is treated as a part of the data segment. The processor state (e.g., registers) is saved by a *setjmp()* system call. The restart of the checkpointed process is simply the reexecution of this executable file on another processor. Li and Fuchs developed a checkpointing scheme for their compiler-assisted checkpoint insertion techniques [16]. Their checkpoint is a data file that contains the data segment and partial stack segment of the checkpointed process. The checkpoint is intended for use in the same shell process on the same machine. Our implementation

uses a checkpoint structure similar to that of Li and Fuchs. In addition to having the complete stack and data segments, our checkpoint also contains a segment for the file I/O output data during that checkpoint interval. The omission of the text segment is possible because the original executable file is already available through NFS. There is no need to transfer the executable file to perform a remote restart.

Two problems have to be overcome for any recovery scheme that uses checkpoint comparison: the remote restartability and comparability of a checkpoint. That is, a task must be able to be restarted from a checkpoint produced on other nodes, and a checkpoint produced on a node must be identical to any checkpoint from any other nodes if both are correct and for the same computation. The former is required for process replication (lookahead execution), while the latter is needed for checkpoint validation.

The uniform virtual memory layout of UNIX in homogeneous machines provides the basis for the restart of a checkpointed process on a remote node. However, some user process information is usually kept in the kernel for efficiency. A checkpoint without this information may not be restartable even for the same kernel. One example is the file I/O information in the file descriptor table in the kernel. When a process terminates or aborts, this information is cleared by the kernel. Restarting a process from a checkpoint without reestablishing this information in another kernel makes a local file descriptor in a user program meaningless.

To make a checkpoint remotely restartable, the user information kept in the kernel has to be extracted during checkpointing and reestablished in the new kernel at restart [14, 15]. A set of library routines was developed for file I/O operations. The library keeps extra data as a part of the checkpoint, such as file name, access mode, and file position, associated with the opened files. During checkpointing, all file buffers are flushed for opened files, and the file positions are updated

and stored in the checkpoint. During a restart, those files are reopened and repositioned according to the previously saved information in the checkpoint. In this manner, the attributes of file I/O can be saved and restored easily across the network. However, the checkpoint may still not be restartable even with the complete information of a user process state, since some state attributes are kernel-dependent. They cannot be saved and carried across kernels (i.e., nodes) in a sensible fashion [14, 15]. Examples are *process group*, *signal* received, the value of real-time clock, and any children that the process may have spawned with *fork()*. Similar to CONDOR and Smith, our current implementation assumes that for restartability a program may not use or depend on those kernel-dependent attributes that have partial information internal to the operating system other than file I/O.

The kernel-dependent attributes also cause checkpoints to be incomparable, even if these checkpoints are all valid. For example, the value of the real-time clock for different kernels may be different, since these clocks are seldom synchronized. The valid checkpoints from the same execution on different nodes may not be the same if the program has these attributes as a part of its memory space. For those kernel-dependent attributes, we enforce the following restrictions to make the checkpoint comparable: we can eliminate the use of variables to store such kernel-specific attributes, or carefully place them in local variables (on the stack) whose scope does not include a checkpoint operation, or clear these variables before checkpointing. Fortunately, most numerical applications seldom use kernel-dependent values except file I/O, and thus meet the restrictions we put on checkpoint restartability and comparability.

B. Experiments

B.1. Benchmark Programs

Four scientific and two SPEC benchmark programs with different checkpoint sizes were selected for our experiments [18]. Program **convlv** is an FFT algorithm that calculates the convolution of 1024 signals with one response, and **ludcmp** is the LU decomposition algorithm that is applied to 100 randomly generated matrices with uniformly distributed sizefrom 50 to 60; **rkf** is the Runge-Kutta-Fehlberg method for solving the ordinary differential equation $y' = x + y$, $y(0) = 2$ with step size 0.25 and error tolerance 5×10^{-7}; **rsimp** is the revised Simplex method for solving the linear optimization problem for the *BRANDY* set from the Argonne National Laboratory. The detailed description of these four benchmark programs can be found in [16]. The **matrix300** and **nasa7** are two SPEC benchmarks: **matrix300** performs various matrix multiplications, including transposes using Linpack routines SGEMV, SGEMM and SAXPY, on matrices of order 300, whereas **nasa7** is a modified version of NASA Ames FORTRAN kernels consisting of seven heavily floating point intensive modules. The original version uses a large memory and generates heavy paging activities on the diskless workstations that lead to a long execution time (44 hours). We have changed some array dimensions so that paging would not delay our experiments (250 K data and about 2 hours of execution).

The checkpoint operations were inserted into these benchmark programs manually. Table 4 summarizes the characteristics for each program with respect to checkpoint size, checkpoint time, checkpoint interval and execution time. Checkpoint size is divided into data segment, stack segment and the file output during the checkpoint interval. Programs **rsimp** and **matrix300** give examples of a large checkpoint. Most applications we examined have checkpoints of size (64-350 K). The stack size is small in all six programs. This is expected for scientific applications in which the calling

Table 4. Overhead Measurements.

Programs Name	# ckp (per run)	ckp_size (data/stack/file) (in bytes)	ckp_time (std. dev.) (in sec)	cmp_time (std. dev.) (in sec)	ckp_interval (std. dev.) (in sec)	exec_time (w/o. ckp) (in sec)
convlv	128	75950 (66196/1554/8200)	0.2172 (0.3411)	0.1608 (8.6302e-3)	13.917 (0.90787)	1809.22 (1781.42)
ludcmp	50	121510 (71708/1550/48252)	0.2408 (3.428e-2)	0.2030 (1.8224e-2)	20.626 (2.1092)	1043.38 (1031.34)
matrix300	150	2219446 (2217652/1794/0)	5.8714 (0.6949)	8.6157 (0.2338)	239.777 (26.729)	37092.88 (36206.30)
nasa7	49	351614 (349788/1826/0)	0.7672 (0.1347)	0.9660 (5.683e-2)	131.46 (28.22)	6611.44 (6573.00)
rkf	88	51777 (46972/1734/3071)	0.1477 (2.563e-2)	0.1492 (7.2498e-3)	29.7202 (1.0840)	2638.58 (2625.58)
rsimp	59	995314 (991676/3638/0)	2.411 (0.3767)	3.8286 (0.21893)	42.8063 (8.6359)	2713.04 (2568.38)

depth is rather limited. The file output size can be large in some applications (e.g., **convlv**).

In Table 4, both *ckp_time* and *cmp_time* do not include the processing time for the file output portion of the checkpoint. For *ckp_time*, the file output portion is already written to disk during execution; thus, it is not necessary to rewrite this portion to the checkpoint. Three variables in a checkpoint are enough to locate this file output portion (file name, starting position and length for each output file). We have found that checkpoint time, comparison time and restart time are highly correlated. Since file I/O operations are the major part of checkpointing (write), checkpoint comparison and restart (read), the overhead costs such as checkpoint time, comparison time and restart time can be expected to be proportional to the size of the checkpoint files.

B.2. Error Detection by Checkpoint Comparison

The effectiveness of checkpoint comparison is studied for the six selected programs. To avoid the interference of run-time error injection with checkpoint comparability, a random bit or word

Table 5. Detectability Experiments.

Program	bit-wise errors					word-wise errors				
	# Errors Detected				#	# Errors Detected				#
	data	stack	file	abort	Missed	data	stack	file	abort	Missed
convlv	68	3	30	0	0	71	0	30	0	0
ludcmp	43	0	58	0	0	37	3	59	2	0
matrix300	101	0	-	0	0	100	0	-	1	0
nasa7	87	0	-	0	14	87	0	-	0	14
rkf	78	1	22	0	0	76	3	22	0	0
rsimp	99	0	-	2	0	98	0	-	2	1

error is injected in the previous checkpoint to model a transient error occurrence during its subsequent checkpoint interval. One task is started from this erroneous checkpoint and another task from the error-free checkpoint. The checkpoints produced by the two tasks after one checkpoint interval are compared. A mismatch indicates a detected error. Table 5 summarizes the results for 101 injected random errors. The number of errors detected is categorized by where the error is detected: the data, stack and the file output segments of the checkpoints. The abortion of the task due to an error in the checkpoint can be treated as a special case of error detection by sending an abortion signal to the voter explicitly.

The errors detected by checkpoint comparison account for the majority of injected errors that occurred (about 98%) for all programs except **nasa7**. If the file output during the checkpoint interval is not included in the checkpoint structure, 22 to 59% of the errors would not be detected (**rkf**, **convlv** and **ludcmp**). Some errors were missed in our experiments. In this case, we have a valid file output during execution and a valid checkpoint at the end; the missed errors are actually masked off and cause no problems with respect to correct executions. This case occurs when an error is in a dead variable and this variable is reinitialized later. A close look at the checkpoint placement for **nasa7** reveals that a new array of about 11% of the total checkpoint size is computed during the checkpoint interval. The 14 missed errors were probably inserted into the new array

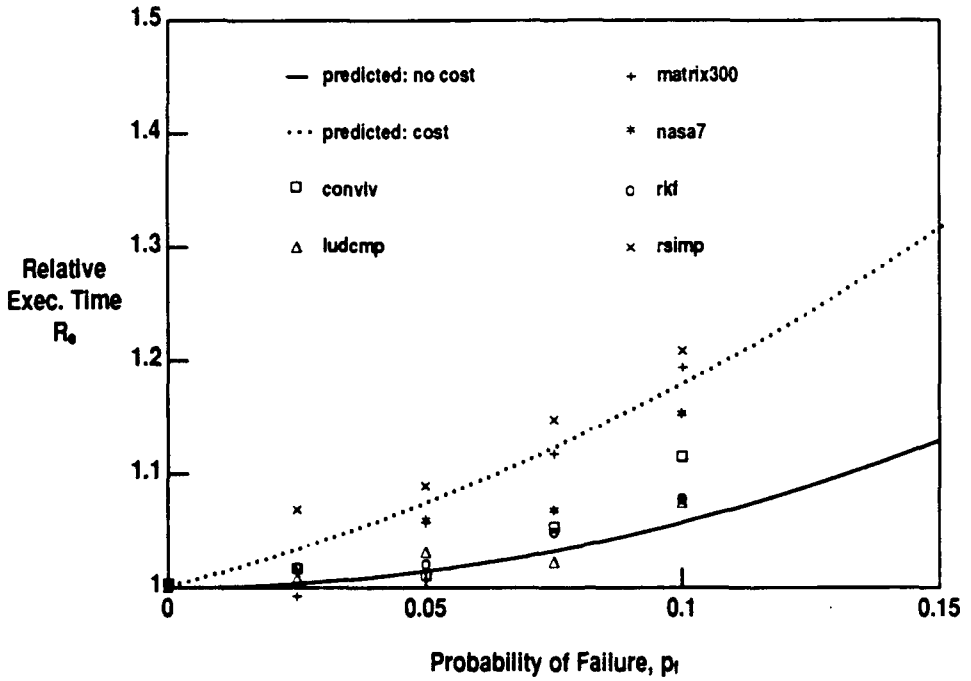

Figure 10. Relative Execution Time.

space and were overwritten during the computation. In sum, the checkpoint structure provided an effective error detection tool for the programs we studied.

B.3. Performance Results

Each program was run five times for each p_f to obtain the average measures. The execution time in our experiments is actually the program response time. It includes the system, user and blocking times. The analytical predictions for the relative execution time, number of processors, and number of checkpoints are also included in Figures 10, 11 and 12 to compare against our experimental results. The data were collected at night to minimize the impact of workload.

In Figure 10, the relative execution time for the programs with a moderate checkpoint size (**ludcmp**, **convlv**, **nasa7** and **rkf**) is close to the analytical zero-overhead prediction (solid curve), since the overheads for those programs is very small compared to their checkpoint intervals. The relative execution time for the programs with large checkpoints (**matrix300** and **rsimp**) fits well with the analytical prediction under a centralized file server (the dotted curve, assuming an overhead

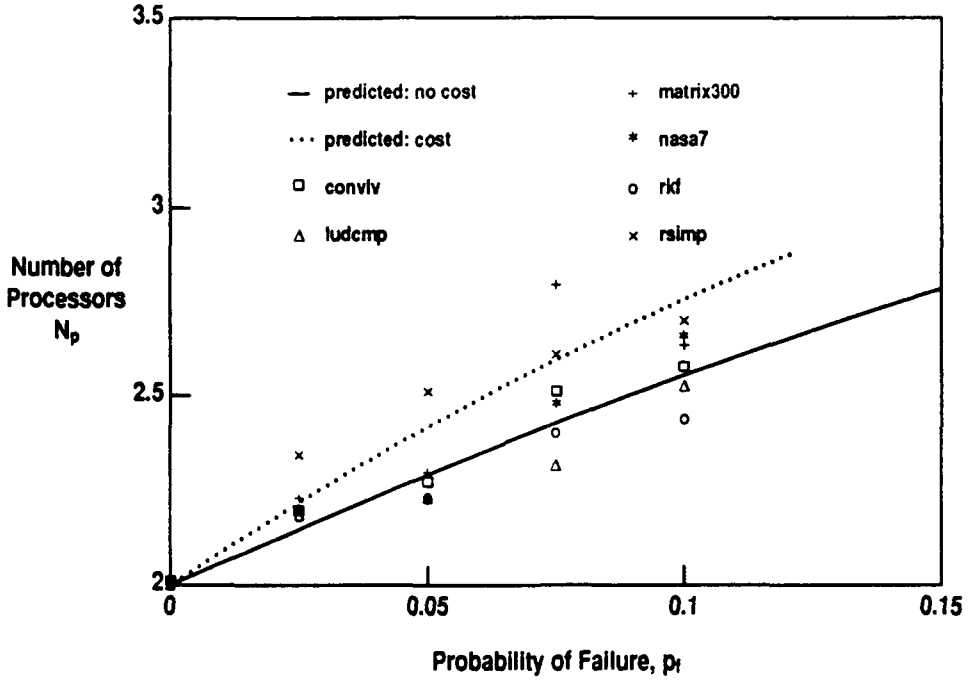

Figure 11. Number of Processors.

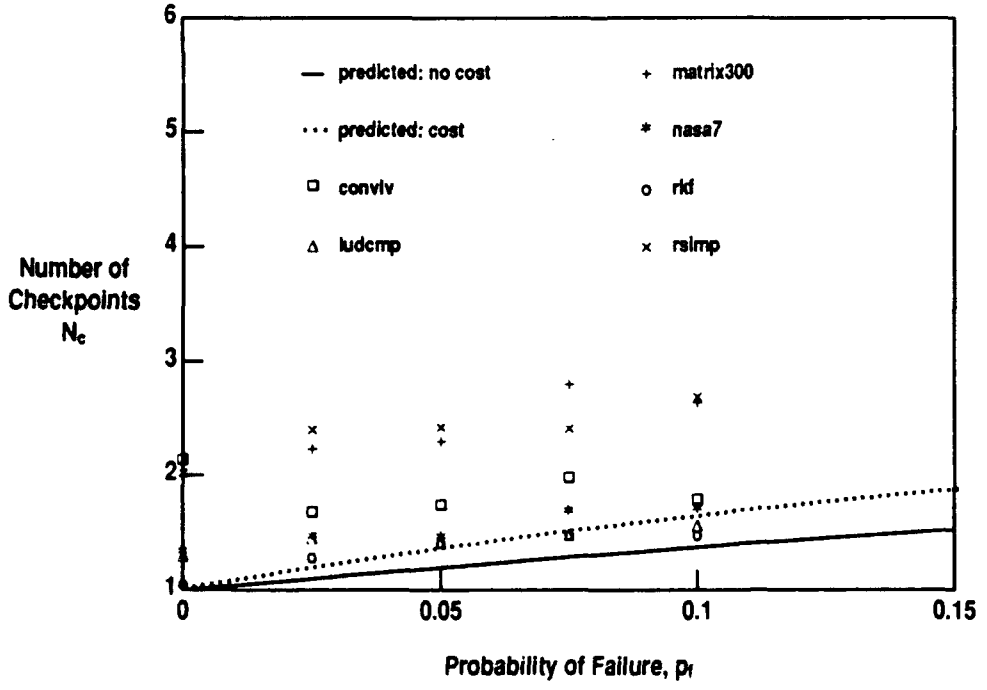

Figure 12. Number of Checkpoints.

level of **rsimp**). This increase in execution time for large checkpoints can be explained by the fact that **matrix300** and **rsimp** are likely to be blocked due to the large file I/O operations during checkpointing and comparison. In fact, the limited speed of the NFS file handling and our use of the file server for managing checkpoints centrally resulted in a performance bottleneck. The paging activities from the replicated processes also contribute to the increase in execution time. The relative execution time increases significantly for high error rates due to the heavy file server activities during checkpointing and comparison of checkpoints. This suggests that a reduction in checkpoint size, an increase in file system speed, or other noncentralized server implementations may improve the execution time over that of our current implementation. The R_e fluctuations in Figure 10 are caused by the uneven network workload distribution at the time of data collection and the small sample size (5) at the low failure rates.

For the p_f we considered, the number of processors used, N_p, is less than the three that TMR requires, although DMR-F-1 uses two more processors momentarily during lookahead/validation operations; N_p is quite insensitive to checkpoint size (Figure 11). The number of checkpoints, N_c, is highly sensitive to the workload and checkpoint size, as a result of the checkpoint accumulation in the file system due to the uneven processor speed (Figure 12), especailly for the programs with large chekpoint sizes.

V. Conclusions

In this paper, we have described a checkpoint-based recovery strategy using optimistic execution and rollback validation for parallel and distributed systems. This approach can reduce rollbacks without depending on specific error-correction knowledge or the standard TMR redundancy. Our recovery schemes (DMR-F-1 and DMR-F-2) can achieve a nearly error-free execution

time with an average redundancy less than that for TMR. In addition, our analysis and experiments have shown that checkpoint comparison time has more impact on performance degradation than restart time. The impact of the centralized file server is also significant for the competing processes during checkpointing, especially for the ones with large checkpoints. Checkpoint comparison was an effective means of error detection and checkpoint validation.

REFERENCES

[1] P. A. Lee and T. Anderson, *Fault Tolerance: Principles and Practice.* Springer-Verlag/Wien, 1990.

[2] P. L'Ecuyer and J. Mallenfant, "Computing optimal checkpointing strategies for rollback and recovery systems," *IEEE Trans. Comput.*, Vol. 37, No. 4, pp. 491–496, April 1988.

[3] S. Toueg and O. Babaoglu, "On the optimum checkpoint selection problem," *SIAM J. Comput.*, Vol. 13, pp. 630–649, Aug. 1984.

[4] C. M. Krishna, K. G. Shin, and Y.-H. Lee, "Optimization criteria for checkpoint placement," *CACM*, Vol. 27, No. 6, No. 6, pp. 1008–1012, Oct. 1984.

[5] A. Duda, "The effects of checkpointing on program execution time," *Information Processing Letters*, Vol. 16, pp. 221–229, 1983.

[6] P. Agrawal, "RAFT: A recursive algorithm for fault-tolerance," *Proc. Int. Conf. Parallel Processing*, pp. 814–821, 1985.

[7] P. Agrawal and R. Agrawal, "Software implementation of a recursive fault-tolerance algorithm on a network of computers," *Proc. 13th Annual Symp. Comput. Arch.*, pp. 65–72, 1986.

[8] N. H. Vaidya and D. K. Pradhan, "Fault-tolerant design strategies for high reliability and safety," Manuscript, Department of Electrical and computer Engineering, University of Massachusetts at Amherst, 1992.

[9] A. Tantawi and M. Ruschitzka, "Performance analysis of checkpointing strategies," *ACM Trans. Comput. Syst.*, Vol. 2, No. 2, pp. 123–144, May 1984.

[10] S. Thanwastien, R. S. Pamula, and Y. L. Varol, "Evaluation of global rollback strategies for error recovery in concurrent processing systems," *Proc. 16th Int. Symp. Fault-Tolerant Comput.*, pp. 246–251, 1986.

[11] Y.-H. Lee and K. G. Shin, "Design and evaluation of a fault-tolerant multiprocessor using hardware recovery blocks," *IEEE Trans. Comput.*, Vol. 33, No. 2, No. 2, pp. 113–124, 1984.

[12] D. J. Taylor and J. P. Black, "A locally correctable b-tree implementation," *Comput. J.*, Vol. 29, No. 3, pp. 269–276, June 1986.

[13] J. Long, "Checkpoint-based forward recovery using lookahead execution and rollback validation in parallel and distributed systems." Tech. Rep. Ph.D. thesis, Center for Reliable and High-Performance Computing, University of Illinois, Urbana, Illinois, 1992.

[14] J. M. Smith, "Implementing remote fork() with checkpoint/restart," *Tech. Committee on Oper. Syst. Newsletter*, Vol. 3, No. 1, No. 1, pp. 15–19, 1989.

[15] M. Litzkow, M. Livny, and M. Mutka, "CONDOR – A hunter of idle workstations," *Proc. 8th int. Conf. Distributed Comput. Syst.*, 1988.

[16] C. C. Li and W. K. Fuchs, "CATCH: Compiler-assisted techniques for checkpointing," *Proc. 20th Int. Symp. Fault-Tolerant Comput.*, pp. 74–81, 1990.

[17] D. J. Taylor and M. L. Wright, "Backward error recovery in a unix environment," *Proc. 16th Int. Symp. Fault-Tolerant Comput.*, pp. 118–123, 1986.

[18] SPEC, *SPEC Newsletter.* Fremont, CA: SPEC, Feb. 1989.

APPENDIX: ANALYTICAL DERIVATIONS

Due to space limitation, we present the analysis only for DMR-F-1 in this appendix. The analysis for DMR-F-2, DMR-B-1, DMR-B-2 and TMR-F is similar [13]. If an error occurs, DMR-F-1 behaves differently in the last checkpoint interval. Since no lookahead execution is possible for this interval, a rollback is always required during recovery. An n-session computation consists of n-1 lookaheadable sessions followed by a rollbackable session. Duda has an excellent analysis for the last rollbackable session [5]. Besides, the performance degradation contribution of the last rollbackable session is proportional to $\frac{1}{n}$, while that of the n-1 lookaheadable sessions is to $\frac{n-1}{n}$. The approximation of an n-session computation with an n lookaheadable sessions is adequate even for a moderate n. Therefore, our analysis focuses on the situation of an n lookaheadable sessions.

Let T_n be the expected execution time for an n-session (lookaheadable) computation, and Let p_l and p_r be the probabilities of a successful lookahead and rollback in DMR-F-1, respectively. Thus, the expected execution time is then.

$$T_n = \begin{cases} \Delta + t_k + T_{n-1}, & 1 - p_l - p_r \\ \Delta + t_k + t_r + 2.5t_t + T_{n-1}, & p_l \\ 2\Delta + 2t_k + 2t_r + 3t_t + T_n, & p_r \end{cases}$$

or,

$$T_n = (\Delta + t_k) + \frac{p_l}{1-p_r}(t_r + 2.5t_t) + \frac{p_r}{1-p_r}(2\Delta + 2t_k + 2t_r + 3t_t) + T_{n-1}.$$

Solving this equation with the initial condition, $T_0 = 0$, we have,

$$\begin{aligned} T_e &= T_n = n(\Delta + t_k) + \frac{np_l}{1-p_r}(t_r + 2.5t_t) + \frac{np_r}{1-p_r}(2\Delta + 2t_k + 2t_r + 3t_t), \\ R_e &= \frac{T_e}{T_0} = 1 + \frac{2p_r}{1-p_r} + \frac{p_l + 2p_r}{1-p_r}\frac{t_r}{\Delta + t_k} + \frac{2.5p_l + 3p_r}{1-p_r}\frac{t_t}{\Delta + t_k}. \end{aligned}$$

The number of processors used is two for the normal execution. It is five during recovery for transient faults and four for permanent faults, since a faulty processor is used in the latter case. In this paper, we analyse only the situation of transient faults, since this gives the worst situation. Let l and r denote $\frac{np_l}{1-p_r}$ and $\frac{np_r}{1-p_r}$ respectively. Therefore,

$$\begin{aligned} \int_0^{T_e} N_p(t)dt &= 2(n-l)(\Delta + t_k) + 2lt_t + 5l(\Delta + t_k + t_r + 1.5t_t) + \\ &\quad +2r(\Delta + t_k + t_r + t_t) + 5r(\Delta + t_k + t_r + 2t_t) \\ &= 2T_e + 3T_0\frac{p_l + p_r}{1-p_r} + 3nt_r\frac{p_l + p_r}{1-p_r} + 3nt_t\frac{1.5p_l + 2p_r}{1-p_r}, \\ N_p &= 2 + 3\frac{p_l + p_r}{(1-p_r)R_e} + 3\frac{p_l + p_r}{(1-p_r)R_e}\frac{t_r}{\Delta + t_k} + 3\frac{1.5p_l + 2p_r}{(1-p_r)R_e}\frac{t_t}{\Delta + t_k}. \end{aligned}$$

There is one (the committed) checkpoint during the normal execution run. Two additional (uncommitted) checkpoints are present during a lookahead/validation operation. At the end of the Δ for the rollback validation, there are eight checkpoints, one committed and seven uncommitted (one for the validation process, two for the normal process pair and four for the lookahead processes). Thus,

$$\begin{aligned}\int_0^{T_e} N_c(t)dt &= n(\Delta + t_k) + 3l(t_r + t_t) + 8l(1.5t_t) + \\ &\quad + r(\Delta + t_k + t_r) + 3r(\Delta + t_k + t_r + t_t) + 8r(2t_t) \\ &= T_e + 2T_0\frac{p_l + p_r}{1 - p_r} + 2nt_r\frac{p_l + p_r}{1 - p_r} + 2nt_t\frac{6.25p_l + 8p_r}{1 - p_r},\end{aligned}$$

$$N_c = 1 + 2\frac{p_l + p_r}{(1 - p_r)R_e} + 2\frac{p_l + p_r}{(1 - p_r)R_e}\frac{t_r}{\Delta + t_k} + 2\frac{6.25p_l + 8p_r}{(1 - p_r)R_e}\frac{t_t}{\Delta + t_k}.$$

If a centralized file server serializes the file accesses, both the restart (t_r) and checkpoint times (t_k) will be increased because of the serialized file accesses generated by checkpointing (write) and restart (read). According to our assumption in Section III, the restart time will be threefold since three processes read the last committed checkpoint file at the same time (the rollback validation and the two lookahead processes). The checkpoint time is $2t_k$ for the normal pair of task replications and $5t_k$ for the lookahead period (four for the lookahead and one for the rollback validation). Thus, the relative execution time with a centralized file server can be shown as

$$R_e(fs) = 1 + \frac{2p_r}{1 - p_r} + \frac{p_l + 5/3p_r}{1 - p_r}\frac{3t_r}{\Delta + 2t_k} + \frac{2.5p_l + 3p_r}{1 - p_r}\frac{t_t}{\Delta + 2t_k} + \frac{p_l + p_r}{1 - p_r}\frac{3t_k}{\Delta + 2t_k}.$$

Following a similar analysis, we can obtain the formulas in Tables 2 and 3.

List of Tables

List of Figures